НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ Олександр Коваль
«___»_____ 2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення розподілених систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Апаратно програмний комплекс моніторингу та управління муніципальними парковками Smart City»

Виконав:
студент IV курсу, групи ТВ-61
Засєка Богдан Анатолійович _____

Керівник:
кандидат наук, доцент
Ковальчук Артем Михайлович _____

Рецензент:
Науковий співробітник
Сірий Олександр Анатолійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення розподілених систем

	ЗАТВЕРДЖУЮ Завідувач кафедри _____ Олександр Коваль (підпис) “___”_____ 2020р.

ЗАВДАННЯ

на дипломну роботу студенту

Засєка Богдан Анатолійвич
(прізвище, ім’я, по батькові)

1. Тема роботи Апаратно програмний комплекс моніторингу та управління муніципальними парковками Smart City

керівник роботи Ковальчук Артем Михайлович андидат наук, доцент
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р.
№ 1168-с

2. Строк подання студентом роботи 5 червня 2020

3. Вихідні дані до роботи JavaScript, React Native, Arduino, NodeJS

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Порівняльний аналіз існуйчих рішенньь, Вибір методі вирішення задачі, Розробка системи

5.Перелік ілюстративного матеріалу: “Актуальність дослідження”, “Архітектура системи”, “Алгоритм роботи системи”, “Схема датчика”, “Використані технології”, “Приклад роботи”, “Висновки”

7. Дата видачі завдання ”11” жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	11.10.2019	
2.	Вивчення та аналіз задачі	11.10.2019-23.12.2019	
3.	Розробка архітектури та загальної структури системи	03.02.2020-04.03.2020	
4.	Розробка структур окремих підсистем	05.03.2020-12.04.2020	
5.	Програмна реалізація системи	13.04.2020-17.05.2020	
6.	Оформлення пояснювальної записки	18.05.2020-07.06.2020	
7.	Захист програмного продукту	14.06.2020	
8.	Передзахист	09.06.2020	
9.	Захист		

Студент _____ ____Засєка Б. А._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ ___Ковальчу А. М._____
(підпис) (прізвище та ініціали,)

Анотація

Мета дипломної роботи полягає у розробці функціональної системи моніторингу і управління муніципальними парковками. На сучасних автостоянках відсутня єдина система для моніторингу місць для паркування. Основною складністю створення єдиної глобальної системи Smart Parking є відсутність стандартизації в програмних і мережевих засобах при побудові систем IoT. Така система дозволить істотно знизити витрати часу і пального при пошуку місця для паркування, адже вона дозволяє проводити моніторинг в реальному часі і бронювати потрібне місце на зазначений час. Для вирішення проблеми в даній роботі використовуються методи аналізу та синтезу, системного аналізу, порівняння, логічного узагальнення результатів. В результаті дипломної роботи було розроблено систему моніторингу муніципальних парковок, яка істотно зменшує час пошуку парковки, дає можливість моніторингу за паркувальними місцями.

Abstract

The purpose of the thesis is to develop a functional system for monitoring and management of municipal parking. Modern parking lots do not have a single system for monitoring parking spaces. The main difficulty in creating a single global Smart Parking system is the lack of standardization in software and network tools when building IoT systems. This system will significantly reduce the cost of time and fuel when searching for a parking space, because it allows you to monitor in real time and book the right place for the specified time. To solve the problem in this work, methods of analysis and synthesis, system analysis, comparison, logical generalization of results are used. As a result of the thesis, a system of monitoring of municipal parking lots was developed, which significantly reduces the time of searching for parking, allows monitoring of parking spaces.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JSON (JavaScript Object Notation) Простий формат обміну даними, зручний для читання як людиною, так і машиною.

СУБД (Система Управління Базами Даних)

API (Application programming interface) Набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.
MVC (Model View Controller) архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

JS (JavaScript)

SQL (Structured query language) мова структурованих запитів

GiST (Generalized Inverted Index) узагальнений інвертний індекс
URI (Universal Resource Identifier) уніфікований ідентифікатор ресурсів

GND (Ground) мінус живлення

ВСТУП

Сучасні автостоянки не мають єдиної системи для моніторингу вільних паркувальних місць. Існуючі системи дуже сильно залежать від людини, це призводить до нерівномірного заповнення стоянок, до збільшення часу на пошук вільних місць, а це у свою чергу призводить до більших викидів CO2 у повітря і великих пробок, що також впливає вцілому на транспортну систему міста. Корпорація IBM провела глобальне дослідження автомобільних парковок (IBM Global Parking Survey)[1]. Дослідження показало, що водії як в розвинених, так і в країнах, що розвиваються стикаються з однією і тією ж незадовільною ситуацією і проблемами паркування. В опитуванні було задіяно 8042 автомобіліста в 20 містах світу на шести континентах. Результати опитування вказують, що водії щодня докладають великих зусиль для пошуку вільного паркувального місця. У минулому році майже 6 з 10 опитаних водіїв принаймні один раз були змушені відмовитися від пошуку місця для тимчасової стоянки свого автомобіля і поїхати в інше місце, і понад чверть респондентів вступали в суперечку з іншими водіями за паркувальне місце.

У дослідженні також відзначається, що поряд зі звичайними пробками більше 30% з них виникають через те, що водії створюють перешкоди на дорозі в процесі пошуку паркувального місця. Неефективні системи організації автостоянок призводять до перевантаженості на дорогах і збільшення викидів вихлопних газів, через них також марнується час водіїв і пасажирів, знижується ефективність праці і губляться економічні можливості.

Проблеми пробок і парковок досягли кризового рівня в усьому світі, вважають в IBM, а викиди вихлопних газів автомобілів ще більше посилюють несприятливу екологічну ситуацію. Всі ці фактори негативно впливають на життя людей в усьому світі, де державні органи, приватний сектор і громадськість шукають нові ефективні

засоби, в тому числі будівництво нових доріг, для пом'якшення і усунення негативних наслідків від завантаженості доріг.

Індустрія парковок унікальна — немає ніякої іншої галузі, де більше 50% доходів надходить від штрафів (для водіїв, які порушують правила паркування). Причина в тому, що ця зріла індустрія навіть 2-3 роки тому ще використовувала застарілу бізнес-модель: з величезною різницею в ціні між відкритою і підземною парковкою, оплатою тільки готівкою, поганими паркувальними знаками і т.д.

І в міру того, як населення міст з плином часу збільшувалася, проблема парковок ставала все більш критичною. Таким чином, поява мобільних платіжних рішень, технологій Інтернету Речей стало ковтком свіжого повітря для галузі, оскільки їх використання дозволяє кардинально поліпшити досвід паркування клієнтів.

Переваги використання рішення для парковки на базі IoT:

- Збільшення доходів від послуг паркування для муніципалітетів і бізнесу.
- Краще управління дорожнім рухом (меньше пробок).
- Зниження викидів CO2.
- Скорочення часу, що витрачається водіями на парковку (час на пошук вільного місця і на оплату паркування).

Задача розробки апаратно програмного комплексу для управління парковками

Завдання даної роботи полягає в розробці апаратно програмного комплекса для управління муніципальними парковками Смарт-сіті. Система розумних парковок це невід'ємна складова розумного міського устрою. Розумні парковки є гарною ілюстрацією до того як зробити Інтернет речей (IoT) частиною нашого повсякденного життя. Розроблена система призначена для використання водіями, та може бути використана для управління та аналізу використання паркувальних місць. Дана система надає водієві доступу до системи Smart Parking через мобільний додаток, щоб знайти та забронювати місце для стоянки в будь-якій зоні для парковки. Це надає можливість зробити оплату чи передоплату за місце для паркування за допомогою кредитної картки. Система дозволяє знайти вільні парковки, повідомити про ймовірність того, що місце стоянки є ще доступним та приймає рішення про резервування та передоплату за таке місце для паркування. Одним із варіантів реалізації є моніторингу стоянки, де кожен паркувальний пункт обладнаний датчиком (фотоапарат або датчик наявності) для виявлення наявності / відсутності транспортних засобів з метою створення карти наявності, яка може бути використана для керування паркуванням, резервуванням та іншими послугам.

Система моніторингу паркувального простору дозволить аналізувати стан паркувальних місць в режимі реального часу, і створить комфортні умови для автовласників в міському середовищі. Ціль роботи є забезпечення контролю паркувального простору за допомогою системи датчиків.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

— Вивчити особливості отримання сигналу від датчиків;

— Розробити ПО для аналізу і обробки сигналів;

— Розробити ПО стану паркувальних місць;

— Провести оцінку обчислювальних ресурсів необхідних для ПО;

— Виконати тестування розробленого програмного забезпечення;

Основною задачею розробленої системи є зменшення часу для пошуку вільних місць, зменшення часу впливає на завантаженість транспортної системи міста та на рівень забруднення повітря. Вхідними даними для задачі є інформація про доступні парковки, яку повинен вносити адміністратор системи та інформація про кількість вільних місць, яка надходить від датчиків які розташовані на парковці. Вихідною інформацією системи є кількість вільних місць. Мета роботи виявити, обгрунтувати і описати переваги інформаційних технологій як інструменту розвитку паркувальних систем. Показати, що з допомогою цього інструменту стає можливим підвищити ефективність і зручність електронних парковок, не витративши на це великої кількості ресурсів.

Об'єктом дослідження є комплекс інформаційних систем для моніторингу паркувального простору. Методи дослідження включають в себе:

— аналіз, порівняння, систематизація та узагальнення даних про існуючих і розроблених способів автоматизації роботи паркувальних комплексів;

— аналіз алгоритмів і підходів, що дозволяють проводити моніторинг за паркувальними місцями;

— експерименти з розпізнавання тестових прикладів;

1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення

Розумна парковка (smart parking) — спеціалізоване місце для парковки автомобілів, створене з використанням датчиків і сучасних технологій для швидкого і зручного пошуку паркувальних місць, забезпечення безпеки і автоматизації процесу паркування автомобіля на стоянку. Створення спеціальних місць для стоянки автомобілів почалося практично одночасно з появою перших автомобілів. Кількість автомобілів стрімко зростає і для вирішення виниклої проблеми обмеженості паркувальних місць стали впроваджувати сучасні технології.

Основним напрямком розвитку є «розумні» датчики паркування. Такі датчики вбудовуються в дорожнє полотно на місця парковок і відстежують зайнято або вільно місце над ними, передаючи дані в загальну систему. Використовуючи мережу таких датчиків, створюється карта парковок, стан якої передається користувачам на вулицях за допомогою спеціальних екранів або мобільного додатка. Існує безліч можливостей збільшити потенціал розумних паркувальних датчиків. Одна з нових концепцій для закритих парковок — навісні датчики, додатково виконують функцію охорони і моніторингу автомобіля. Ще одним напрямком розумних парковок (smart parking) є розробка і впровадження автоматизованих парковок (найчастіше багаторівневих), в яких дії водіїв зведені до мінімуму. Водій заїжджає на спеціальний майданчик / платформу і виходить з машини. Потім платформа сама переносить автомобіль на спеціально відведене, зарезервоване або вільне місце, і повідомляє водієві його номер. Щоб отримати свій транспортний засіб, водієві необхідно авторизуватись і ввести даний номер на спеціальному табло або пульті управління, після чого платформа також самостійно спустить автомобіль на майданчик.

1.2 Огляд існуючих аналогів

1.2.1 Parkeon

Parkeon присутній в більш ніж 4000 містах і мегаполісах в 60 країнах світу, пропонуючи інноваційні інтелектуальні транспортні рішення і паркувальні рішення [2]. Послідовність дій відповідно до малюнком 1:

Рисунок 1.1 — схема роботи сервісу Parkeon

1 в'їзд:

— Машина зупиняється навпроти в'їзду, прямо перед шлагбаумом;

— Сенсор розпізнає присутність машин, а відповідна камера автоматично розпізнає і записує номер машини, дату і час в'їзду;

— Дистанційна система управління отримує дані, надані цією камерою і додає їх в базу даних.

Якщо клієнт вже зареєстрований, то додається також інформація про членство і дата резервування;

— Шлагбаум на в'їзді піднімається, дозволяючи водієві потрапити на парковку. Шлагбаум автоматично опускається, коли машина проходить через сенсор безпеки;

2 виїзд:

— Машина зупиняється на виїзді, прямо перед шлагбаумом;

— Сенсор розпізнає присутність машини, а камера перевіряє держ. номер на відповідність інформації про платіж, або набору машин в базі даних;

— Система управління валідує коректний платіж, або дозвіл на парковку без платежу, щоб дозволити виїзд;

— Шлагбаум на виїзді піднімаємося, дозволяючи водієві покинути парковку;

— Шлагбаум автоматично опускається, як тільки машина проїжджає через сенсор безпеки;

1.2.2 Мікком AS101 ProPark

Принцип роботи полягає в точному визначенні розташування вільних і зайнятих місць і / або в підрахунку кількості в'їхали і виїхали машин. Світлова індикація кожного паркувального місця і інформаційні табло вказують водію вільні місця і оптимальний маршрут до них відповідно до малюнком 2. Система забезпечує оперативний і постійний контроль завантаженості з наданням персоналу всієї

необхідної інформації, управління світлофорами, табло та шлагбаумами [3]. Система виконана на базі комплексу безпеки AS101 Pro і володіє властивою комплексу високою надійністю, зручністю і простотою експлуатації.

Рисунок 1.2 — Приклад реалізації Мікком AS101 ProPark

В рамках даного проекту було створено ряд нових пристроїв (Ультразвукові датчики присутності автомобіля відповідно до малюнком 3, виносні індикатори

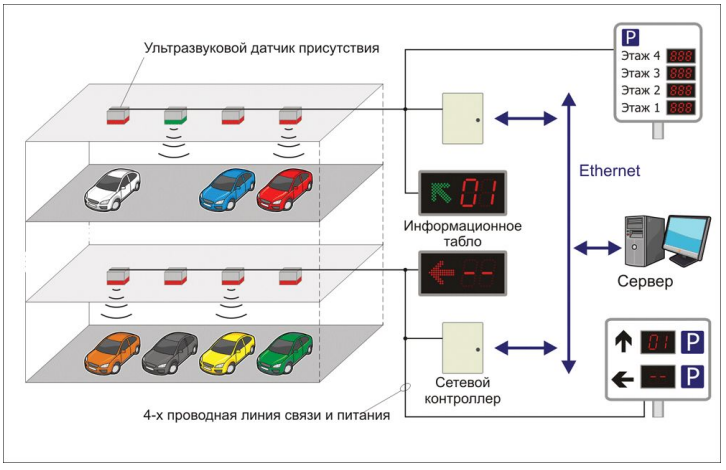


Рисунок 3 — схематичне зображення Мікком AS101 ProPark

зайнятості місця, інформаційні табло, датчики в'їзду / виїзду, лічильники-суматори проїхали автомобілів) і ПО.

1.2.3 Аксіома Груп "Розумна парковка»

Основним завданням проекту є впровадження программно аппаратного комплексу, що дозволяє в режимі реального часу відслідковувати стан кожної паркувального місця. Рішення базується на технології компанії Nedap AVI, яка, в свою чергу, розробила рішення SENSIT. Суть рішення полягає у впровадженні в кожне паркувальне місце датчика, який за допомогою електромагнітного і інфрачервоного сенсорів визначає наявність або відсутність транспортного засобу. далі по радіоканалу датчики передають інформацію на базові станції, які акумулюють інформацію по групі місць і передають її на керуючий сервер. Розроблене компанією Аксіома Груп програмне забезпечення дозволяє систематизувати отриману інформацію з датчиків, побудувати аналітичну звітність щодо зайнятості, оборотності паркувальних місць, передавати інформацію в навігаційні системи, а також на світлодіодні табло, розташовані в місті. Все обладнання, яке використовується в вуличних умовах, володіє сертифікацією не нижче IP65 і повністю адаптовано під кліматичні умови м.Москва. Робота по проекту велася в тісній зв'язці зі службами міста, включаючи Департамент Капітального Ремонту, ЦКУ Адміністратор Московського паркувальні Простору. Система введена в експлуатацію і вже сьогодні дозволяє автолюбителям і службам міста отримувати інформацію по завантаженості паркувальних місць на вулицях міста в межах Бульварного кільця [4].

2. ЗАСОБИ РОЗРОБКИ

Для клієнтської частини було обрано фреймворк React Native, для серверної частини продукту було обрано Node.js — JavaScript оточення побудоване на JavaScript рушієві Chrome V8. Основним фактором вибору технологій було швидкість розробки та мультиплатформеність. В якості хмарного провайдера було обрано Heroku, головним плюсом цього провайдера є те, що він підтримує найпопулярніші мови програмування, а також надає безкоштовний акаунт і базу даних. Для датчика було обрано Sharp IR Sensor який підключається до апаратно обчислювальної платформи Arduino, для зв'язку датчика і сервера було використано Ethernet mini модуль. В якості бази даних було обрано PostgreSQL.

2.1 React Native

React Native — це програма з відкритим кодом для створення додатків для Android та iOS з використанням власних можливостей платформи React та платформи додатків. Завдяки React Native ви використовуєте JavaScript для доступу до API вашої платформи, а також для опису зовнішнього вигляду та поведінки вашого інтерфейсу за допомогою компонентів React. Принцип роботи React Native практично ідентичний React, за винятком того, що React Native не маніпулює DOM за допомогою Virtual DOM. Він запускається у фоновому процесі (який інтерпретує JavaScript, написаний розробниками) безпосередньо на кінцевому пристрої та спілкується з нативною платформою через серіалізацію, асинхронний та пакетний міст.

Реактивні компоненти обертають наявний нативний код та взаємодіють із нативними API через декларативну парадигму інтерфейсу користувача та JavaScript. Це дає змогу розробляти нативну програму для цілих нових команд розробників, а також дозволяє діючим командам працювати набагато швидше.

React Native не використовує HTML або CSS. Натомість повідомлення з потоку JavaScript використовуються для маніпулювання власними видами. React Native також дозволяє розробникам писати нативний код такими мовами, як Java для Android та Objective-C або Swift для iOS, що робить його ще більш гнучким.

2.2 React

ReactJS - це бібліотека з відкритим кодом, що базується на компонентах, і відповідає тільки за шар перегляду застосування. Його підтримує Facebook.

ReactJS використовує віртуальний DOM-механізм для заповнення даних (переглядів) у HTML DOM. Віртуальний DOM працює швидко до того, що він змінює лише окремі елементи DOM замість того, щоб щоразу перезавантажувати повний DOM.

Додаток React складається з декількох компонентів, кожен з яких відповідає за виведення малого шматка багаторазового використання HTML. Компоненти можна вкладати в інші компоненти, щоб збільшити складність програми, побудовані з простих будівельних блоків. Компонент також може підтримувати внутрішній стан - для прикладу, компонент TabList може зберігати змінну, що відповідає відкритій вкладці. React дозволяє писати компоненти за допомогою доменної мови під назвою JSX. JSX дозволяє писати компоненти, що використовують HTML, змішуючи події JavaScript. React внутрішньо перетворює це у віртуальний DOM та в кінцевому рахунку виведе HTML для споживача.

React реагує на зміни стану компонентів швидко та автоматично, щоб передати компоненти в HTML DOM, використовуючи віртуальний DOM. Віртуальний DOM - це уявлення в пам'яті фактичного DOM. Виконуючи більшу частину обробки всередині віртуального DOM, а не безпосередньо в DOM браузера, React може діяти швидко та лише додавати, оновлювати та видаляти компоненти, які змінилися з моменту останнього циклу візуалізації.

2.3 Node js

Середовище виконання Node.js включає все, що потрібно для виконання програми, написаної на JavaScript. Node.js використовує подію, що не блокує модель вводу / виводу, що робить її легкою та ефективною.

Node.js має наступні властивості:

— асинхронна модель виконання запитів;

— неблокуючий ввід/вивід;

— система модулів CommonJS;

— рушій JavaScript Google V8;

Встановлення та видаленням модулів в Node.js займається утиліта під назвою npm(node package manager).

Середовище Node.js використовується для написання високопродуктивних мережевих систем, написаних на мові програмування JavaScript. Платформа використовується також у створенні клієнтських застосунків та серверних програм, для роботи із серверними скриптами В платформі використовується розроблений компанією Google рушій V8.

Обрана для Node.js асинхронна модель запуску коду дозволяє обробляти велику кількість паралельни запитів. Дана модель визначає обробник викликів (callback) та запускає його в обробку подій в неблокуючому режимі. Для підтримки мультиплексування з'єднань використовується дані технології epoll, kqueue, /dev/poll і select. Для взаємодії з даними модулями операційної системи використовується бібліотека libuv, для створення пулу нитей (thread pool) використовується libeio, для не блокуючих DNS-запитів інтегровано c-ares. Всі системні виклики, що спричиняють блокування, виконуються всередині пулу ниток і потім, як і обробники сигналів, передають результат своєї роботи назад через неіменовані канали (pipe).

2.4 Heroku

Heroku — хмарний провайдер PaaS-платформа, який підтримує більшість мов програмування. Програми, які запускаються на Heroku, зазвичай мають унікальний домен (як правило, "applicationname.herokuapp.com"), який використовується для маршрутизації HTTP-запитів до правильного контейнера додатків, або dyno. Кожен з dyno розповсюджується на "дино-сітку", яка складається з декількох серверів. Сервер Git Heroku обробляє натискання репозиторію додатків від дозволених користувачів. Усі послуги Heroku розміщені на платформі хмарних обчислень EC2 від Amazon. Для кожної програми виділяється кілька незалежних віртуальних процесів, які називаються «dynos». Вони розподілені по спеціальній віртуальній сітці («dynos grid»), яка складається з декількох серверів. Heroku також має систему контролю версій Git.

2.5 Arduino

Arduino (Ардуино) — апаратна обчислювальна платформа, головним компонентами якої є плата мікроконтролера з елементами вводу/виводу та середовище розробки Processing/Wiring на мові програмування, що є схожою та спрощеною підмножиною C/C++. Arduino може використовуватися як для створення автономних інтерактивних об'єктів, так і підключатися до програмного забезпечення, яке виконується на комп'ютері (наприклад: Processing, Adobe Flash, Max/MSP, Pure Data, SuperCollider). Інформація про плату (рисунок друкованої плати, специфікації елементів, програмне забезпечення) знаходяться у відкритому доступі і можуть бути використані тими, хто воліє створювати плати власноруч. Плата Arduino складається з мікроконтролера Atmel AVR, а також елементів обв’язки для програмування та інтеграції з іншими пристроями. На багатьох платах наявний лінійний стабілізатор напруги +5В або +3,3В. Тактування здійснюється на

частоті 16 або 8 МГц кварцовим резонатором. У мікроконтролер записаний завантажувач (bootloader), тому зовнішній програматор не потрібен. Плати Arduino дозволяють використовувати значну кількість виводів мікроконтролера як вхідні/вихідні контакти у зовнішніх схемах. Ардуіно і Ардуіно-сумісні плати спроектовані таким чином, щоб їх можна було при необхідності розширювати, додаючи в пристрій нові компоненти («shields»). Ці плати розширень підключаються до Ардуіно за допомогою встановлених на них штирових роз'ємів. Існує ряд уніфікованих плат, що допускає конструктивно жорстке з'єднання процесорної плати та плат розширення в стопку через штирові лінійки. Крім того, випускаються плати зі зменшеним (наприклад, Nano, Lilypad) і спеціальним (для задач робототехніки) форм-фактором.

Інтегроване середовище розробки Arduino це багатоплатформний додаток на Java, що включає в себе редактор коду, компілятор і модуль передачі прошивки в плату. Середовище розробки засноване на мові програмування Processing та спроектоване для програмування новачками, незнайомими близько з розробкою програмного забезпечення. Мова програмування аналогічна мові Wiring. Загалом, це C++, доповнений деякими бібліотеками. Програми обробляються за допомогою препроцесора, а потім компілюються за допомогою AVR-GCC.

Програми Arduino пишуться на мові програмування C або C++. Середовище розробки Arduino поставляється разом із бібліотекою програм «Wiring». Користувачам необхідно визначити лише дві функції для того, щоб створити програму, яка буде працювати за принципом циклічного виконання:

— setup(): функція виконується лише раз при старті програми і дозволяє задати початкові параметри

— loop(): функція виконується періодично, доки плата не буде вимкнена

2.6 Postgres

PostgreSQL — об'єктно-реляційна система керування базами даних (СКБД). У PostgreSQL є підтримка індексів наступних типів: B-дерево, хеш, R-дерево, GiST, GIN. При необхідності можна створити нові типи індексів. У даній роботі крім розповсюджених індексів було використано GiST індекс для швидкого пошуку найближчих точок. GiST (англ. Generalized Search Tree, Узагальнене пошукове дерево) — структура індексу, яка є узагальненою різновидом R-tree і надає стандартні методи навігації по дереву і його поновлення (розщеплення і видалення вузлів). GiST є збалансованим (по висоті) дерево, кінцеві вузли (листя) якого містять пари (key, rid), де key - ключ, а rid - покажчик на відповідний запис на сторінці даних. Внутрішні вузли містять пари (p, ptr), де p - це якийсь предикат (використовується як пошуковий ключ), що виконується для всіх спадкових вузлів, а ptr - вказівник на інший вузол в дереві. Для цього дерева визначені базові методи SEARCH, INSERT, DELETE, і інтерфейс для написання користувальницьких методів, якими можна управляти роботою цих (базових) методів.

Метод SEARCH управляється функцією Consistent, що повертає 'true', якщо вузол задовольняє предикату, метод INSERT - функціями penalty, picksplit і union, які дозволяють оцінити складність операції вставки в вузол, розділити вузол при переповненні і перебудувати дерево при необхідності, метод DELETE знаходить лист дерева , що містить ключ, видаляє пару (key, rid) і, якщо потрібно, за допомогою функції union перебудовує батьківські вузли [1].

GiST є прямим індексом, використовуваним повнотекстових пошуком СУБД PostgreSQL. Це означає, що для кожного вектора tsvector, що описує всі лексеми документа, створюється сигнатура, що описує, які з лексем входять в даний tsvector. Принцип роботи схожий з бітовими індексами, проте є і відмінності. Продемонструємо на прикладі: нехай лексемі w1 співставлена сигнатура 001000,

лексемі w2 - 000010. Тоді документа, який містить тільки лексеми w1 і w2, буде сполучати сигнатура 001010 (001000 | 000010). На відміну від бітових індексів, відображення лексем в сигнатури неоднозначно, тобто можливе існування лексеми w3 з сигнатурою 001000. Це дозволяє значно економити на розмірі індексу, але вимагає вторинної перевірки на повний збіг лексем запиту і документа. Варто також звернути увагу, що в разі, якщо лексема запиту має сигнатуру, наприклад, 000001, то документ з сигнатурою 001010 однозначно її не містить, не дивлячись на неоднозначність відображення лексем.

Перевага GiST в його швидкості створення і розмірі індексу в порівнянні з GiN (в 3 рази), тому він краще для динамічно постійно оновлюваних даних. Для статичних даних або які рідко оновлюються краще GiN індекс (він в 3 рази швидше здійснює пошук).

2.7 JSON Web Token (JWT)

Для ідентифікації користувача в системі було обрано технологію під назвою JSON Web Token. JSON Web Token (JWT) - це стандарт токена доступу на основі JSON, стандартизованого в RFC 7519[8]. Використовується для верифікації тверджень. JWT складається з трьох частин: заголовка, вмісту і підпису. Заголовок (Header) це JSON елемент, який описує до якого типу токену належить даний і які методи шифрування використовувались.

Поле	Назва	Значення
typ	Type	Описує медіатип IANA Medientyp токену. В даному випадку JWT завжди мають медіатип application/jwt.
cty	Content Type	Це поле потрібне, коли JWT вміщає в себе інший JWT. Тоді воно встановлюється в JWT. В інших випадках це поле пропускається.
alg	Algorithm	Описує використаний алгоритм шифрування. Зазвичай використовують HMAC з SHA-256 (HS256) або RSA з SHA-256 (RS256). Можна також не використовувати

		жодного шифрування, вказавши none, але це не рекомендується. Можливі значення вказуються в стандарті JSON Web Encryption (JWE) RFC 7516.

Таблиця 2.1 - структура JWT заголовку

Вміст (англ. Payload) складається з елемента JSON який описує твердження.

Поле	Назва	Значення
iss	Issuer	У "iss" визначається довіритель, який видав JWT. Обробка цього рядка, як правило, специфічна для заявки. Значення "iss" - це чутливий до регістру рядок, що містить StringOrURI значення. Використання цієї вимоги опційно.
sub	Subject	чутливий до регістру рядок чи URI.
aud	Audience	масив чутливих до регістру рядків або URI, який є списком одержувачів даного токена. Коли приймаюча сторона отримує JWT з даними ключем, вона повинна перевірити наявність себе в іменах одержувачів - інакше проігнорувати токен.
exp	Expiration Time	час в форматі Unix Time, що визначає момент, коли токен стане недійсним.
nbf	Not Before	на противагу ключу exp, цей час в форматі Unix Time, що визначає момент, коли токен стане дійсним.
iat	Issued At	час в форматі Unix Time, що визначає момент, коли токен був створений.
jti	JWT ID	рядок, що визначає унікальний ідентифікатор даного токена.

Таблиця 2.2 - структура вмісту

Як правило, при використанні JSON-токенів в клієнт-серверних додатках реалізована наступна схема:

1. Клієнт проходить аутентифікацію в додатку (наприклад, з використанням логіна і пароля).
2. У разі успішної аутентифікації сервер відправляє клієнту access- і refresh-токени.

3. При подальшому зверненні до сервера клієнт використовує access-токен. Сервер перевіряє токен на валідність і надає клієнту доступ до ресурсів.
4. У разі, якщо access-токен стає недійсним, клієнт відправляє refresh-токен, у відповідь на який сервер надає два оновлених токена.
5. У разі, якщо refresh-токен стає не дійсним, клієнт знову повинен пройти процес аутентифікації (п. 1).

Access-токен - це токен, який надає доступ його власнику до захищених ресурсів сервера. Зазвичай він має короткий термін життя і може нести в собі додаткову інформацію, таку як IP-адресу запитувача даного токена. Refresh-токен - це токен, що дозволяє клієнтам запитувати нові access-токени після закінчення їх часу життя. Даний токен зазвичай видається на тривалий термін.

JWT має ряд переваг над кукою:

- При використанні куки сервер повинен зберігати інформацію про видані сесіях, в той час як використання JWT не вимагає зберігання додаткових даних про видані токенах: все, що повинен зробити сервер - це перевірити підпис.
- Сервер може не займатися створенням токенів, а надати це зовнішньому сервісу.
- У JSON-токенах можна зберігати додаткову корисну інформацію про користувачів. Як наслідок - більш висока продуктивність. У разі с куки іноді необхідно здійснювати запити для отримання додаткової інформації. При використанні JWT ця інформація може бути передана в самому токені.
- JWT надає можливість одночасного доступу до різних доменів і сервісів.

3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Система управління парковками повинна давати можливість переглядати найближчі парковки до користувача, бронювати місця, а також слідкувати за місцями на парковці.

Система управління парковками складається з трьох частин:

1. Додаток на телефон
2. Сервер
3. Датчик, який знаходиться на парковці

На рисунку 1 зображена схема взаємодії всіх підсистем. Основний сценарій системи

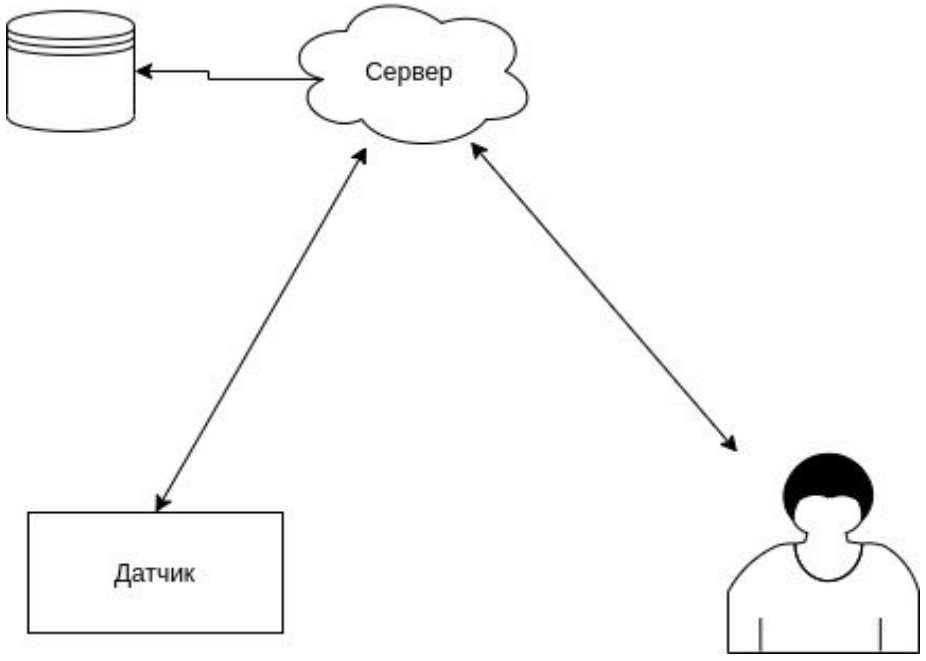


Рисунок 3.1 — архітектура системи

полягає у тому що юзер посилає свої координати на віддалений сервер, він у свою чергу посилає запит у базу даних яка знаходить найближчі парковки по даній координаті.

Коли користувач займає довільну парковку, датчик надсилає запит у якому знаходиться ідентифікатор який записаний в базі даних, по цьому ідентифікатору сервер дістає з таблиці датчиків координату зайнятого місця. Також у системі присутні дані про координати користувачів, це зроблено для того, щоб під часа того коли сервер отримає сигнал від датчика він зміг знайти з усіх юзерів найближчого до даного датчика. Для цього використовується функціонал бази даних під назвою GiST індекс, який дозволяє використовувати алгоритм пошуку найближчих сусідів. У системі присутній такий випадок коли на паркувальне місце заїжджає не авторизований користувач, у такому випадку ми маркуємо дане паркувальне місце як зайняти невідомим автомобілем.

3.1 Додаток на телефон

Додаток для телефону виконаний на базі фреймворку React Native, що дозволило написати один додаток на дві платформи. Основною функцією програми є відображення всіх вільних парковок у певному радіусі, бронювання на певний період, а також прокладання маршруту до вибраної парковки. У додатку було використано паттерн проектування який має назву Model-View-ViewModel. На рисунку 3.2 зображена схема роботи даного патерна Цей шаблон широко використовується у проектуванні клієнтських додатків. Model-View-ViewModel (MVVM) - це структурна модель дизайну, яка розділяє об'єкти на три різні групи:

- Моделі містять дані програми. Зазвичай це структури або прості заняття.
- Перегляди відображають візуальні елементи та елементи керування на екрані. Вони, як правило, підкласи UIView.
- Переглянуті моделі перетворюють інформацію про модель у значення, які можуть відображатися на поданні. Зазвичай вони заняття, тому їх можна передавати як посилання.

Вхідною точкою для додатку є файл App.js, в цьому файлі знаходиться головний клас в якому додаток у системи запрошує доступ до геолокації користувача. Перед відображенням компонента програмний код дістає токен користувача з мобільного сховища.

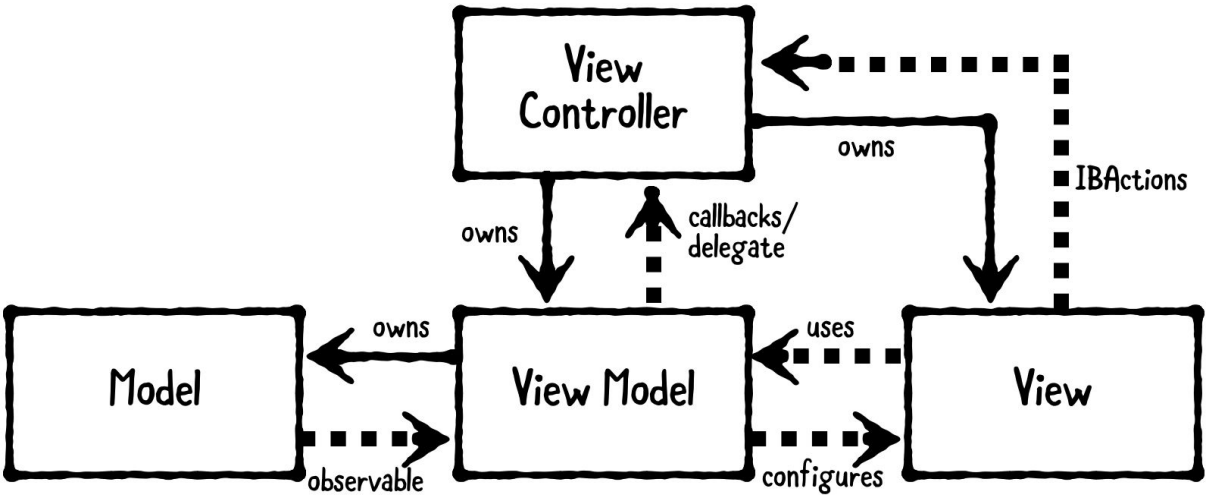


Рисунок 3.2 - структура MVVM паттерн

Це сховище є безпечним і зберігає всю інформацію доки, користувач не видалить додаток. Якщо токена в додатку не виявилося, програма ідентифікує користувача як не авторизованого. Далі відображається форма входу, якщо користувач не авторизований, або завантажується мапа. Під час завантаження мапи додаток отримує геодані від користувача, за допомогою цих даних мапа відображає поточне розташування користувача, також ці дані використовуються для запиту за найближчими парковками для користувача. Після отримання даних від сервера додаток проходячись по цьому масиву відображає на мапі маячки чорного кольору з кількістю вільних місць. Під час запиту за парковками усі дані про них не дістаються, оскільки це дуже сильно економить трафік користувача. Кожен маячок на карті це елемент для якого визначений обробник на клік користувача по ньому. При проходженні по масиву парковок для кожного маячка закріплюється його ідентифікатор, цей ідентифікатор передається в функцію обробник події клік. У свою чергу ця функція зберігає даний ідентифікатор у замикання і повертає іншу

функцію, яка має доступ до цього ідентифікатора. Замикання це підпрограма яка виконується у середовищі яке містить одну або більше зв'язаних змінних. Ця підпрограма має доступ до змінних у бідь-який час її виконання. Під час кліку викликається визначена функція, яка з замикання дістає ідентифікатор парковки, на яку натиснув користувач, далі передає цей ідентифікатор у функцію, яка викликає вбудований метод для роботи з HTTP, під назвою fetch. Даний метод повертає детальну інформацію про обрану парковку, ця інформація включає в себе:

- адрес парковки
- фото
- кількість вільних місць
- ціна за годину

Ця додаткова інформація передається у компонент під назвою CardInfo. Його функція полягає у відображенні детальної інформації користувачеві. Цей компонент складається з головного тега під назвою div, це є умовою фреймворка, кожен компонент повинен повертати тільки один елемент.

Головний тег включає в себе картинку обраної парковки, деяку інформацію, а також у нижній частині цього елемента відображається дві кнопки. За кожною кнопкою закріплений певний функціонал. Для першою кнопки це бронювання. Під час натисканні на цю кнопку програмний код викликає функцію обробник, який отримує ідентифікатор парковки, ця функція викликає HTTP запит з шляхом, який закріплений за функціоналом бронювання. Після цієї дії додаток перезавантажує інформацію про всі парковки, оскільки змінилася кількість вільних місць. Інша кнопка відповідає за функціонал прокладання маршруту. Для цього було використано сервіс від Google під назвою Google Maps. Викликана функція передає параметр у бібліотеку яка використовується для роботи з даним сервісом, у свою чергу дана бібліотека відображає на мапі найкоротший шлях до даної точки. При кліку на пусте місце картка з додатковою інформацією про парковку зникає.

3.2 Сервер

Архітектура серверної частини системи створена з допомогою одного за найуживаніших патернів проектування MVC (Model-View-Controller). Використання даного патерна дає змогу чітко і лаконічно розподілити логіку роботи серверного додатку.

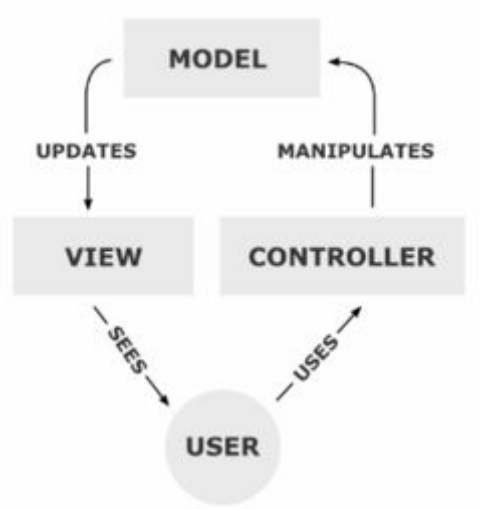


Рисунок 3.3 — Приклад роботи паттерну MVC

Архітектурний шаблон модель–вигляд–контролер (MVC) поділяє додаток на три окремі незалежні, але взаємопов'язані, між даними частинами розподіляється окремий незалежний функціонал. Функцією Моделі(Model) є збереження даних у сховище, а також однією з головних функцій компонента є забезпечення взаємодії з цим сховищем даних. Функцією вигляда (View) є представлення даних, які приходять від моделі, користувачеві. Функцією контролера (Controller) є керування вище переглянутими компонентами, даний компонент отримує сигнал, який виникає на дію користувача, обробивши його цей сигнал передається у модель. У даній системі контролерами є функції які знаходяться у файлі handlers. Кожен контролер приймає об'єкт який є відображенням запиту від зовнішнього клієнта. Модель є

відображення поведінки застосунка та є центральним, інтерфейс користувача не має прямоє взаємодії з моделю. Правилами застосунку та логікою системи є модель, в даному випадку уся бізнес логіка системи знаходиться у папці storage(сховище). Ця папка розподілена на файли, кожен з яких є відображення бізнес моделі яка використовується у даній системі, а саме користувачу відповідає файл user.ts, файл parking.ts відповідає за бізнес правила для взаємодії з парковками. Пряме керування даними є головним обов'язком моделі. Будь яке представлення інформації від системи це і є вигляд, тобто графіки діаграми, можна віднести до цього компоненту. В деяких системах можуть існувати безліч виглядів, наприклад гістограма для керівництва компанії й таблиці для бухгалтерії.

Модель є незалежної від вводу і виводу даних у системі, її головною метою є інкапсулювання ядра системи. Вигляд може мати кілька взаємопов'язаних областей, наприклад різні таблиці і поля форм, в яких відображаються дані. Дії користувача викликають у системі деякі сигнали які відстежує контроллер, що і є однією з функції описаного компонента. Пов'язані дії у системі зазвичай структуруються гляхом групування кодових елементів в окремий клас. Для прикладу візьмемо типовий функціонал проекта, а саме реєстрацію, цей контроллер об'єднується в клас зі схожою назвою і він містить групу методів які пов'язані з управлінням аккаунта користувача, туди входить зміна паролю, редагування профілю, авторизація та реєстрація.

Події які зареєстровані в системі перетворюється у різні запити, які спрямовуються в компоненти моделі або компонентам які відповідальні за відображення даних кінцевому користувачеві. Для незалежного використання різних компонентів, даний патерн відокремлює модель від відображення інформації. Таким чином, зміни від користувача внесені до моделі даних, які подані декількома візуальними компонентами, будуть автоматично скориговані відповідно до змін, що відбулися.

Зважаючи на це і було обрано архітектуру MVC. Основою додатку було створено контроллер який реагує на всі дії користувача у системі. Тобто він розподіляє абсолютно усю логіку роботи сайту. Зважаючи на розмір додатку прийнятним рішення було обрати даний підхід для реалізації архітектури даної системи. У більших системах контролери поділяються на більшу кількість логічних видів, кожен з який є обробником для певної дії, в окремій частині системи..

Задача сервера у даній системі є керуванням доступом до даних, транформацією їх для кінцевого клієнта та знаходження найближчих парковок до даної точки. Для цього було використано алгоритм пошуку найближчої точки до даної точки. Звичайне B-дерево не підходить для такого типу даних, так як для точок не визначені оператори порівняння. З цією задачею добре справляється GiST індекс в СУБД PostgreSQL. GiST — скорочення від “generalized search tree”, це збалансоване дерево пошуку, складається з вузлів-сторінок. Вузли складаються з індексованих записів. Кожен запис листового вузла містить предикат (логічний вираз) і посилання на рядок таблиці (TID). Індексовані дані повинні задовольняти цьому предикату. Пошук в дереві GiST використовує спеціальну функцію узгодженості (consistent) — одну з функцій, яка визначаються інтерфейсом, і реалізована по-своєму для кожного підтримуваного сімейства операторів. Пошук проводиться в глибину: алгоритм в першу чергу намагається дістатися до якогось листового вузла. Це дозволяє по можливості швидко повернути перші результати.

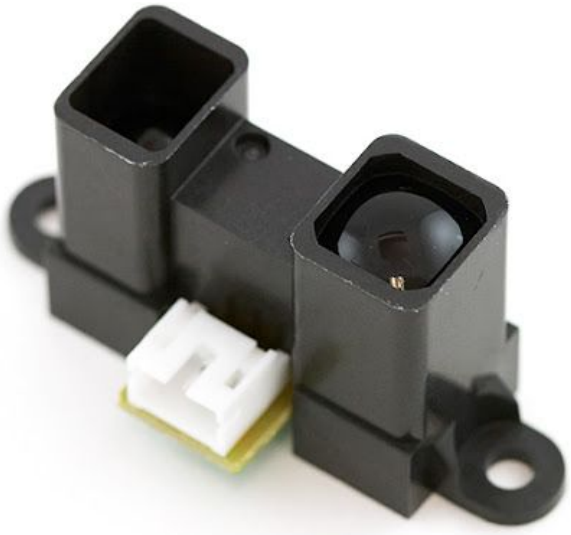
3.3 Датчик

У даній системі датчик виконує роль сигналу про від'їзд або приїзд автомобіля на паркувальне місце. Для аналізу присутності автомобіля було розглянути такі види датчиків:

1. Магнітний
2. Тиску

3. Інфрачервоний

Магнітний датчик мав би вбудовуватися в землю під паркувальним місцем, а однією з головних вимог до системи це легкість її інтегрування в існуючі парковки. Також мінусом цього датчика, це те що не всі авто зроблені з металу на який реагує магніт. Датчик тиску також відпав оскільки його теж потрібно вбудовувати в землю і площа його дії була малою. Вибір зупинився на інфрачервоному датчику відстані, який кріпиться перед паркувальним місцем. На рисунку 3.4 зображено інфрачервоний датчик GP2Y0A02YK0F призначений для вимірювання відстані до перешкод від 20 до 150 см.



Рисунку 3.4 — інфрачервоний датчик відстані

Напруга на аналоговому виході датчика відповідає відстані до перешкоди. Датчик складається з позиційно-чутливого детектора, інфрачервоного світлодіода, оптичних лінз та схеми обробки сигналів. Застосування методу тріангуляції та схеми обробки сигналів знижує вплив відображаючих властивостей об'єктів, кольору,

температури навколишнього середовища і тривалості експлуатації на показання датчика. Вихід датчика (жовтий дріт) підключається до будь-якого аналогового входу Arduino. Напруга живлення 5 В постійного струму подається на Vcc (червоний дріт) і GND (чорний дріт) датчика. Вихід датчика відстані Sharp обернено пропорційний - із збільшенням відстані це зменшується і зростає повільно. Точний графік між відстанню і виходом наведено на рисунку 3.5. У датчиків, відповідно до типу, є межа вимірювання, в межах якої вихід датчика є надійним.

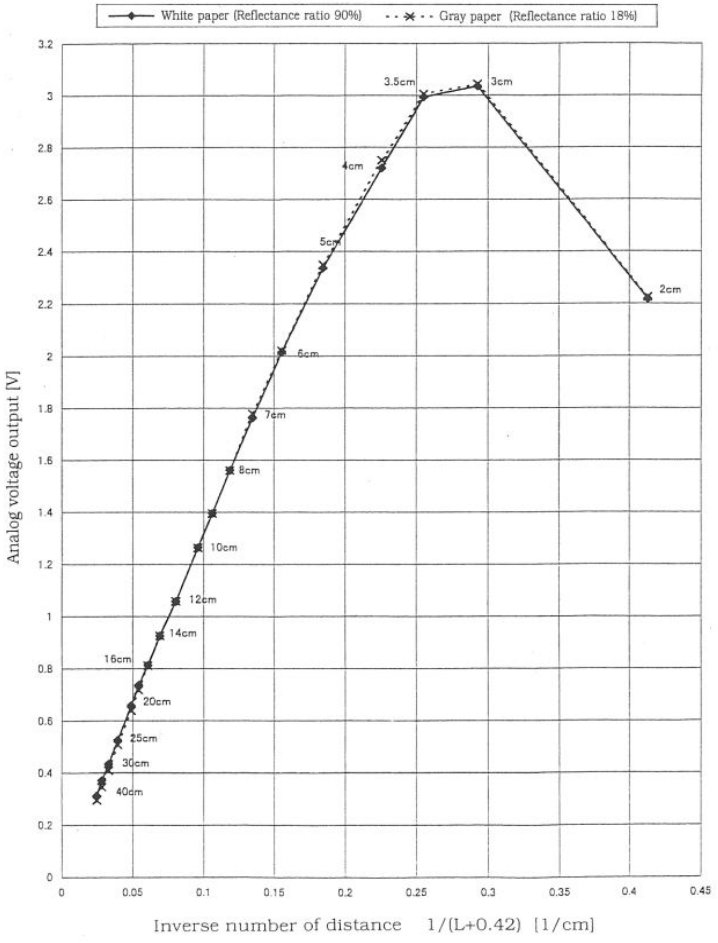


Рисунок 3.5 - графік залежності напруги на виході від відстані

Вимірювання максимально реального відстані обмежують два аспекти: зменшення інтенсивності відбивається світла і неможливість PSD реєструвати зміна місця

розташування відображеного маленького променя. При вимірі сильно віддалених об'єктів, вихід датчика залишається приблизно таким же, як і при вимірюванні максимально віддалених відстаней. Мінімально вимірювана відстань обмежена особливостями датчика Sharp, а саме - вихідна напруга на певній відстані (в залежності від датчика: 4-20 см) починає різко падати при зменшенні відстані. По суті це означає, що одному значенню вихідної напруги відповідає два відстані. Для запобігання проблеми потрібно уникати занадто близького наближення об'єктів до датчика.

Для передачі даних на сервер було вирішено обрати модуль mini Ethernet. Модуль працює в локальній мережі TCP \ IP і в мережі internet. Використовуючи апаратні можливості модуля ENC28J60 і програму мікроконтролера можна створити простий веб-сервер. Модуль пов'язує МК через інтерфейс SPI з мережею TCP \ IP. Управління приладами автоматики через інтернет підключення до ENC28J60 може проводиться одночасно або по черзі декількома операторами з різних персональних комп'ютерів або мобільних пристроїв. Цей модуль розбражений на рисунку 3.6 і має такі характеристики:

- Напруга живлення 3,14 - 3,45 В;
- Струм, мА: максимальний 250; номінальний 170;
- Інтегрований MAC і 10 Base-T PHY;
- Підтримка одного 10 Base-T порту з автовизначенням полярності і корекцією;
- Автовирівнювання генерації контрольної суми;
- Відкидання помилкових пакетів;
- Програмована функція повтору передачі при помилку;
- Програмовані функції Padding і генерування CRC;
- Програмована функція фільтрації помилкових пакетів;
- Тактова частота SPI до 20 МГц.

В якості апаратно обчислювальної платформи було обрано Arduino Mega 2560. Arduino Mega 2560 - це пристрій на основі мікроконтролера ATmega2560.

Рисунок 3.6 - модуль mini Ethernet ENC28J60

У його склад входить все необхідне для зручної роботи з мікроконтролером: 54 цифрових входу / виходу (з яких 15 можуть використовуватися в якості ШІМ-виходів), 16 аналогових входів, 4 UART (апаратних приймача для реалізації послідовних інтерфейсів), кварцовий резонатор на 16 МГц , роз'єм USB, роз'єм живлення, роз'єм ICSP для внутрішньо схемного програмування і кнопка скидання. Для початку роботи з пристроєм досить просто подати живлення від AC / DC-адаптера або батарейки, або підключити його до комп'ютера за допомогою USB-кабелю. Характеристики плати:

- Мікроконтролер ATmega2560

- Робоча напруга 5В
- Напруга живлення (рекомендована) 7-12В
- Напруга живлення(ліміт) 6-20В
- Цифрові входи/виходи 54 (з яких 15 можуть використовуватися в якості ШІМ-виходів
- Аналогові входи 16
- Максимальний струм одного вивіда 40 мА
- Максимальний вихідний струм вивіда 3.3V 50 мА
- Flash-пам’ять 256 КБ із якої 8 КБ використовується завантажувачем
- SRAM 8 КБ
- EEPROM 4 КБ
- Тактовая частота 16 МГц

Програмний код для мікроконтролера завантажується з допомого студії Arduino IDE. Для взаємодії з Ethernet модулем було використано бібліотеку з відкритим кодом EtherCard.

4. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для роботи даної системи знадобиться телефон на базі Android/iOS, зі стабільним інтернет з'єднанням. Для стабільної і коректної роботи паркувального датчику потрібно непреривне інтернет з'єднання, для живлення у повнорозмірних плат Arduino (Duemilanove, Uno, Mega, Leonardo і т. П.) є лінійний регулятор напруги і роз'єм підключення блоку живлення 5,5мм / 2,1 мм (зовнішній / внутрішній діаметр). До нього можна підключати джерело живлення постійного струму з напругою від 9 до 12 Вольт.

Після успішного встановлення програми на телефон, користувачу буде запропонована форма логіну, яка зображена на рисунку 4.1

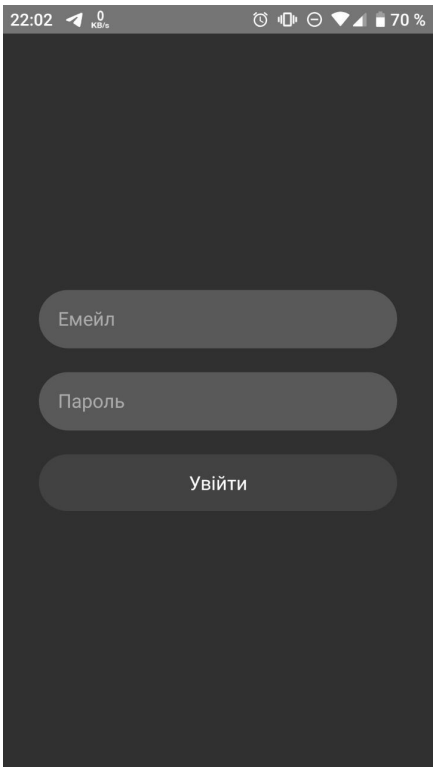


Рисунок 4.1 — форма логіну

Усі дані про користувачів вносяться адміністраторами системи через спеціальну систему. Після успішного логіну телефон користувача надсилає координати на сервер, після чого він отримує десять найближчих парковок. Всі парковки відмічені на карті чорними маячками на яких позначена кількість вільних місць, парковки без доступних місць не відображаються. На рисунку 4.2 зображена мапа, вона є повністю інтерактивною, користувач може віддаляти зближати і переміщати її.

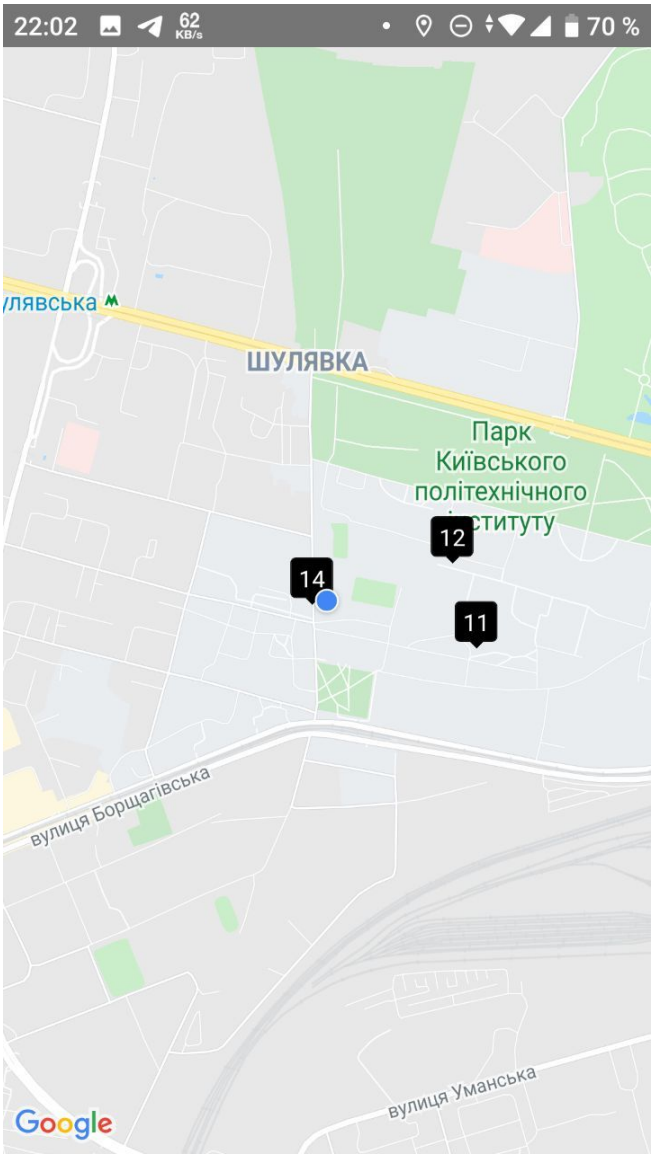


Рисунок 4.2 — інтерактивна мапа з парковками

Синім маячком зображений користувач, при русі чи повороті маячок реагує на це і повторює. При кліку на чорний маячок відображається додаткова інформація про

дану парковку. На рисунку 4.3 зображена додаткова інформація, а саме, назва паркувального майданчика, його адреса, а також зображення для кращого орієнтира

для водія. Під адресою відображена максимальна місткість парковки та ціна за годину користування. У самому низі розташовані дві кнопки “Забронювати” та

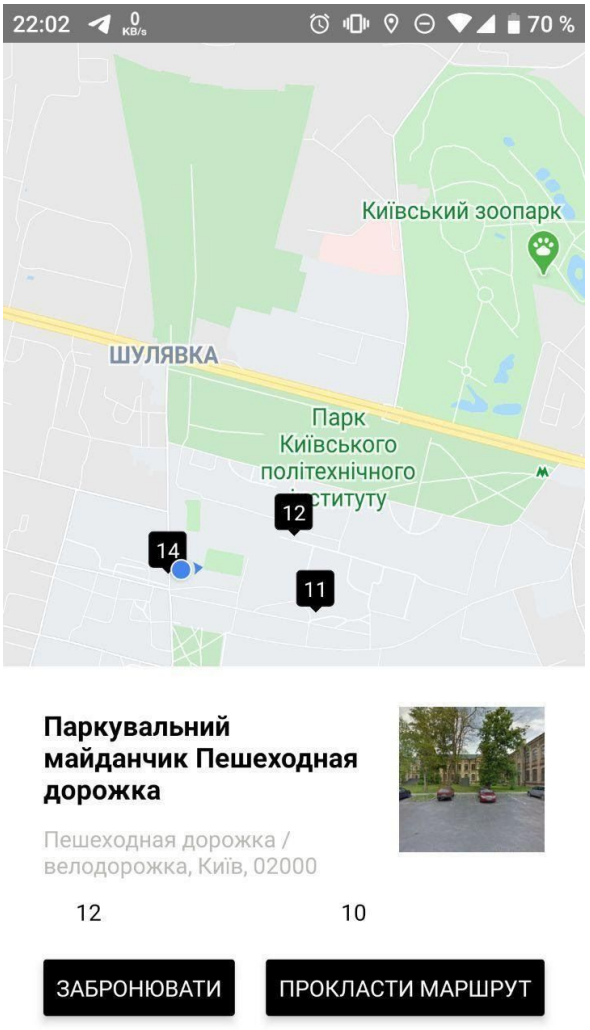


Рисунок 4.3 — додаткова інформація про парковку

“Прокласти маршрут”. Перша кнопка виконує бронювання вибраного місця на певний період часу, після закінчення броні місце повертається у резерв вільних. Інша кнопка прокладає оптимальний маршрут з урахування усіх пробок да дорожніх ситуацій. На рисунку 4.4 зображена приклад маршруту до майданчика. Після повторного входу користувач заново переходить на екран з мапою, оскільки при

першому вході додаток запам'ятовує ідентифікатор користувача за яким сервер розуміє хто це є. При русі автомобіля додаток надсилає координати, що допомагає зрозуміти який автомобіль зайняв конкретне місце. При паркуванні користувачу буде надіслане сповіщення з даними про паркування.

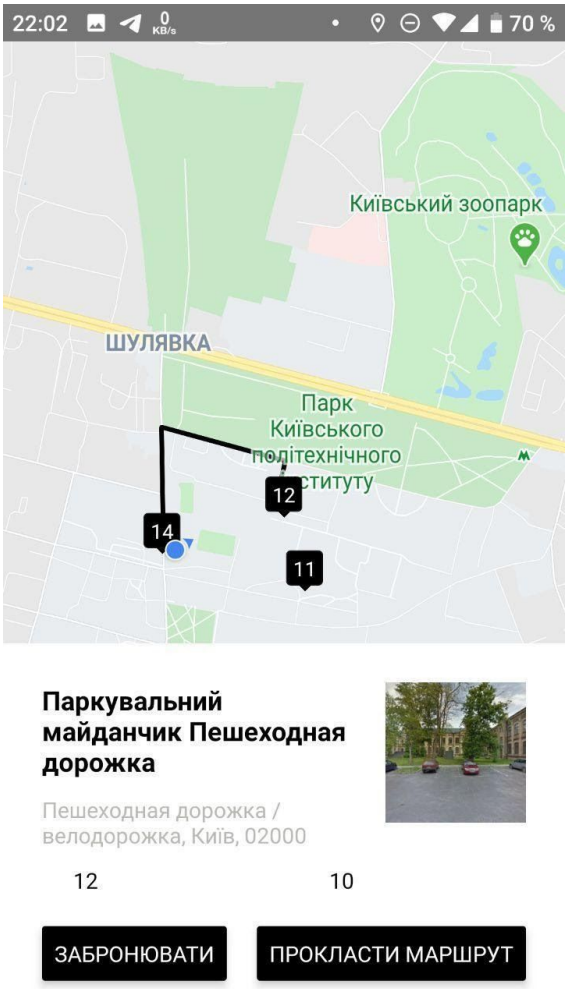


Рисунок 4.4 — приклад прокладання маршруту

Після закінчення стоянки користувач отримує сповіщення про оплату. Усі дані про майданчики вносяться за допомогою адмін частини системи. На головному екрані зображена карта, при кліку на яку відкривається форма для заповнення даних.

ВИСНОВОК

Згідно індивідуального завдання була розроблена системи для автоматизації та адміністрування паркувальних місць. Під час написання даної роботи було ознайомлено із загальною проблематикою управління паркувальними місцями, проаналізовано елементну базу сенсорів, розроблено алгоритм для вирішення задачі та на його основі створено систему. Проаналізувавши існуючі рішення було прийняте рішення зробити систему яка задовольняє всім поставленим вимогам. Також, окремим розділом, було коротко розглянуто існуючі рішення та проаналізовано особливості кожної системи.

В ході даної роботи було розроблено систему яка складається з таких частин: датчика, який закріплюється на паркувальному місці, мобільний додаток, який надає змогу взаємодіяти з розробленою системою, та сервер, який займається підрахунком та зберіганням усією інформації яка використовується у системі.

Джерела

1. IBM Global Parking Survey: Drivers Share Worldwide Parking Woes [Електронний ресурс] — Режим доступу: https://www-03.ibm.com/press/us/en/pressrelease/35515.wss
2. Smart Parking Systems [Електронний ресурс] — Режим доступу: https://smartparkingsystems.com/en/
3. Офіційний сайт компанії Микком [Електронний ресурс] — Режим доступу: http://miccom.ru/products/pm-about/pm1
4. «Розумні парковки» - новий підхід до вирішення проблеми паркінгу в містах. Офіційний сайт компанії Акіома [Електронний ресурс] — Режим доступу: http://www.aksioma-group.ru/proekty/sistema-monitoringa-parkovochnykh-mest-v-predelakh-bulvarnogo-koltsa-moskvy/
5. Документація по технології React Native [Електронний ресурс] — Режим доступу: https://reactnative.dev/
6. Документація по хмарному сервісу Heroku [Електронний ресурс] — Режим доступу: https://www.heroku.com/platform
7. Документація по експлуатації інфрачервоного датчика IR Sharp GP2Y0D02YK0F [Електронний ресурс] — Режим доступу: https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0d02yk_e.pdf
8. Стандарт JSON Web Token (JWT) [Електронний ресурс] — Режим доступу: https://tools.ietf.org/html/rfc7519

Додаток А

Апаратно програмний комплекс моніторингу і управління парковками

Специфікація

УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6132_20Б

Аркушів 1

Київ - 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6132_20Б	Пояснювальна записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6132_20Б 12-1	Arduino.ino, handlers.ts, MapElement.tsx	Основні компоненти
УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6132_20Б 13-1	Додаток В.docx	Опис програмного модуля

Додаток Б

Апаратно програмний комплекс моніторингу і управління парковками

Текст програми

УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6132_20Б 12-1

Аркушів 7

Київ - 2020

```
// модуль датчика, який выдправляэ сигнали на сервер
#include <EtherCard.h>
#define ir A6 // пін на який подається напруга від інфрачервоного датчика
#define MIN_DISTANCE 40 // дистанція в см при якій датчик відправляє сигеал

// ethernet interface mac address, must be unique on the LAN
static byte mymac[] = { 0x74,0x69,0x69,0x2D,0x30,0x31 }; // ethernet мак адреса, має
бути унікальною всередині локальної мережі

byte Ethernet::buffer[700];
static uint32_t timer;
const char website[] PROGMEM = "http://whispering-ravine-61460.herokuapp.com"; //
адреса сервера

int getDistance() {
 // функція яка переводить напругу в дистанцію
  int voltFromRaw = analogRead(ir);
  return 9462.0 / (voltFromRaw - 16.92);
}

// викликається коли клієнт отримав відповідь від сервера
static void my_callback (byte status, word off, word len) {
  Serial.println(">>>");
  Ethernet::buffer[off+len] = 0;
  char* fullResponse = (char*) Ethernet::buffer + off;
  Serial.println(fullResponse);
}

void setup() {
  Serial.begin(9600);
  // ініцілізація Ethernet модуля
  if (ether.begin(sizeof Ethernet::buffer, mymac, SS) == 0)
    Serial.println(F("Failed to access Ethernet controller"));
  if (!ether.dhcpSetup()) // отримання DHCP інформації від роутера
    Serial.println(F("DHCP failed"));

  if (!ether.dnsLookup(website)) // отримання айпі від DNS сервісу
    Serial.println("DNS failed");

  // вивід DHCP інформації
  ether.printIp("IP:  ", ether.myip);
  ether.printIp("GW:  ", ether.gwip);
  ether.printIp("DNS: ", ether.dnsip);
  ether.printIp("SRV: ", ether.hisip);
}
void loop() {
  ether.packetLoop(ether.packetReceive());
  if (millis() > timer) {
      // отримуємо дистанцію до обєкта
    int puntualDistance = getDistance();
    timer = millis() + 2000;
    if (puntualDistance < MIN_DISTANCE) {
        // якщо дистанція менша чим задане число, то відправляємо сигнал на сервер
    ether.browseUrl(
      PSTR("/parking-signal?id=123456"),
      "",
      "",
      my_callback
    );
```

```
    }
    ether.persistTcpConnection(true);
  }
}
import jwt from 'jsonwebtoken';
import bcrypt from 'bcrypt'
import Boom from '@hapi/boom';

import { Context } from "server/typings/common";
import { SECRET_KEY } from './index';
import {
  getNearestParks,
  getDetails,
  bookParking,
  getSensor
} from "./storage/parking";
import {
  getUserByEmail,
  createUser,
  insertUserPosition,
  getNearestUser,
  createUserStatisitc,
  getStatisticBySensor,
  endParking,
  getUserStatus
} from './storage/user';

const saltRounds = 10;

export const getParkPlaces = async (request: Context) => {
  // повертає користувачеві найближчі парковки
  const { x, y } = request.query;
  const parsedX = parseFloat(typeof x == "string" ? x : x[0]);
  const parsedY = parseFloat(typeof y == "string" ? y : y[0]);

  return await getNearestParks({ x: parsedX, y: parsedY }, request.data["number"]);
};

export const bookPlace = async (request: Context) => {
// забронювати місце
  return await bookParking(parseInt(request.params.id));
};

export const getPark = async (request: Context) => {
// отримати додаткову інформацію про парковку
  return await getDetails(parseInt(request.params.id));
};

export const login = async (request: Context) => {
// логін користувача
  const { email, password } = request.data;
  const user = await getUserByEmail(email);
  if (!user) {
    throw Boom.unauthorized('unauthorized');
  }
  const match = await bcrypt.compare(password, user.password);
  if (match) {
    const session = jwt.sign({ id: user.id, email: user.email }, SECRET_KEY);
    request.res.cookie('session', session);
```

```
    return {status: 'OK', session};
  } else {
    throw Boom.unauthorized('unauthorized');
  }

}

export const registrate = async (request: Context) => {
// реєстрація
  const { email, password } = request.data;

  const salt = await bcrypt.genSalt(saltRounds);
  const hashedPassword = await bcrypt.hash(password, salt);

  await createUser(email, hashedPassword);

  return {status: 'OK'};
}

export const tracking = async (request: Context) => {
// збереження координат користувача
  const { point } = request.data;

  const user_data = jwt.decode(request.headers['x-token'], { json: true });
  if (user_data) {
    const user_id = user_data['id'];
    await insertUserPosition(user_id, point);
  }

  return {status: 'OK'};
}

export const parkingSignal = async (request: Context) => {
// опрацювання сигналу від датчика на парковці
  const { id } = request.query;
  let sensor;
  if (typeof id === "object") {
    sensor = await getSensor(id[0]);
  } else {
    sensor = await getSensor(id);
  }

  const statistic = await getStatisticBySensor(sensor.id);

  if (statistic) {
    // end parking
    await endParking(statistic.id);
  } else {
    const user = await getNearestUser(sensor.point);
    if (user) {
      await createUserStatisitc(user.id, sensor.id);
    }
  }

  return {status: 'OK'};
}
```

```
export const userStatus = async (request: Context) => {
// повертає інформацію про користувача
  const user_data = jwt.decode(request.headers['x-token'], { json: true });
  if (user_data) {
    const data = await getUserStatus(user_data['id']);
    return data;
  }

  return {}
}


// елемент мапа
import React, {useEffect, useState} from 'react';
import {
  StyleSheet,
  View,
  ActivityIndicator,
  Dimensions,
  Easing,
  Animated,
} from 'react-native';

import Geolocation, { GeolocationResponse } from '@react-native-community/geolocation';
import MapView, {PROVIDER_GOOGLE, Marker} from 'react-native-maps';
import MapViewDirections from 'react-native-maps-directions';
var PushNotification = require("react-native-push-notification");

import ParkingMarker from './ParkingMarker';
import CardInfo from './CardInfo';
import {fetchParks, getParkingDetails, bookParking, getUserStatus} from '../api/parks';
import {getCurrentPosition, sendCurrentPosition} from '../api/geolocation';

import {GOOGLE_MAPS_APIKEY} from '../secrets';

// отримати розміри екрану, для коректного відображення мапи
const {width, height} = Dimensions.get('window');
const LATITUDE_DELTA = 0.0122;
const ASPECT_RATIO = width / height;
const LONGITUDE_DELTA = LATITUDE_DELTA * ASPECT_RATIO;
const CARD_HEIGHT = 230;

const MapElement = () => {
  const [currentPosition, setCurrentPostition] = useState<GeolocationResponse>();
  const [parks, setParks] = useState<NearestParks>([]);
  const [selectedParking, setSelectedParking] = useState<ParkPlace>();
  const [destinationParking, setDestinationParking] = useState<ParkPlace>();
  const [bookedParking, setBookedParking] = useState<number>();
  useEffect(() => {
    let showNotification = false;
// запускаємо таймер
    const timerId = setInterval(async () => {
      // отримати інформацію про користувача
      const data = await getUserStatus();
      if (data.datetime && !data.end_datetime) {
        const message = `Ви зайняли парковку в ${new
Date(data.datetime).toLocaleTimeString()}`;
        if (!showNotification) {
          // відобразити нотифікацію з інформацією про парковку
```

```
          PushNotification.localNotification({message});
          showNotification = true;
        }
      }

      if (data.datetime && data.end_datetime) {
        // випадок коли користувач закінчив стоянку
        const duration = (
          new Date(data.end_datetime).valueOf() - new Date(data.datetime).valueOf()
        ) / 60000
        const message = `Ви покинули парковку в ${new
Date(data.end_datetime).toLocaleTimeString()}. Простояли ${duration} хвилин.`;
        PushNotification.localNotification({message});
        clearInterval(timerId); // зупиняєсо таймер
      }

    }, 10000);
  }, []);

  useEffect(() => {
    (async () => {
      const position = await getCurrentPosition(); // отримати поточні координати
користувача
      setCurrentPostition(position);
      try {
        // отримуємо від сервера найближчі парковки до даної координати
        const parks = await fetchParks(position.coords);
        setParks(parks);
      } catch (e) {
        console.log(e.message);
      }
    })();
  }, [bookedParking]);

  const [fadeAnim] = React.useState(new Animated.Value(-200));
  // слідкуємо за позицією користувача і відправляємо її на сервер
  Geolocation.watchPosition(
    point => {
      sendCurrentPosition(point);
    },
    () => {},
    { distanceFilter: 0, enableHighAccuracy: true, maximumAge: 0}
  );

  const openCard = (parkingId: number) => async () => {
    Animated.timing(fadeAnim, {
      toValue: 0,
      duration: CARD_HEIGHT,
      easing: Easing.sin,
    }).start();
    setSelectedParking(undefined);
    // запрошуємо додаткову інформацію для вибраної парковки
    const parking = await getParkingDetails(parkingId);
    setSelectedParking(parking);
  };
  const closeCard = () => {
    Animated.timing(fadeAnim, {
      toValue: -CARD_HEIGHT,
      duration: CARD_HEIGHT,
```

```
    easing: Easing.sin,
  }).start();
};
if (!currentPosition) {
  return (
    <View style={styles.container}>
      <ActivityIndicator size="large" color="black" />
    </View>
  );
};

const onBook = async (parking: ParkPlace) => {
  const id = await bookParking(parking.id);
  setBookedParking(id);
}

return (
  <View style={styles.container}>
    <MapView
      onPress={() => closeCard()}
      showsMyLocationButton={true}
      provider={PROVIDER_GOOGLE}
      initialRegion={{
        latitude: currentPosition.coords.latitude,
        longitude: currentPosition.coords.longitude,
        latitudeDelta: LATITUDE_DELTA,
        longitudeDelta: LONGITUDE_DELTA,
      }}
      loadingEnabled={true}
      loadingIndicatorColor="#666666"
      loadingBackgroundColor="#eeeeee"
      showsUserLocation={true}
      style={styles.map}>
        <>
        {parks.map(park => (
          <ParkingMarker
            key={park.id}
            onPress={openCard(park.id)}
            coordinate={{
              latitude: park.coordinate.x,
              longitude: park.coordinate.y,
            }}
            freeAmount={park.freePlaces}
          />
        ))}
        {destinationParking && (
          <MapViewDirections
            optimizeWaypoints={true}
            mode="DRIVING"
            strokeWidth={3}
            origin={currentPosition.coords}
            destination={{
              latitude: destinationParking.coordinate.x,
              longitude: destinationParking.coordinate.y,
            }}
            apikey={GOOGLE_MAPS_APIKEY}
          />
        )}
      </>
```

```
      </MapView>
      <Animated.View style={{...styles.cardInfo, bottom: fadeAnim}}>
        {selectedParking ? (
          <CardInfo
            parking={selectedParking}
            onCreateDirection={(parking) => setDestinationParking(parking)}
            onBook={onBook}
          />
        ) : (
          <ActivityIndicator size="large" color="black"/>
        )}
      </Animated.View>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    ...StyleSheet.absoluteFillObject,
    justifyContent: 'center',
    alignItems: 'center',
  },
  map: {
    ...StyleSheet.absoluteFillObject,
  },
  cardInfo: {
    height: CARD_HEIGHT,
    position: 'absolute',
    width: '100%',
    justifyContent: "center"
  },
  button: {
    width: 80,
    paddingHorizontal: 12,
    alignItems: 'center',
    marginHorizontal: 10,
    backgroundColor: '#242424',
  },
  buttonContainer: {
    flexDirection: 'row',
    marginVertical: 20,
    backgroundColor: 'transparent',
  },
});

export default MapElement;
```

Додаток В

Апаратно програмний комплекс моніторингу і управління парковками

Опис програми

УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6132_20Б

Аркушів 1

Київ - 2020

АНОТАЦІЯ

Система являє собою самодостатню систему для управління і моніторингу паркувальних місць. Дана система призначена як і для користувача парковок, так і для персоналу який обслуговує їх. У функціонал системи входить: перегляд найближчих парковок, бронювання, перегляд детальної інформації про парковку, можливість прокладання оптимального маршруту. Також система автоматично веде облік усіх зайнятих і вільних парковок. Для персоналу система надає можливість відслідковувати статистичну інформацію, можливість моніторингу активних користувачів у системі.

Серверний модуль було розроблено на базі технології NodeJS, що дало можливість швидкого прототипування системи. Мобільний додаток було на базі технології React Native.

ЗМІСТ

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Система складається з трьох підсистем: сервер, мобільний додаток, датчик. Дана система розв'язує проблему управління та моніторингу міських парковок. Дана задача вирішена завдяки технології React Native для мобільного додатку, NodeJS на стороні сервера, апаратна частина завдяки технології Ардуїно.

Система має такий функціонал:

- відображення найближчих парковок.
- бронювання парковки
- прокладання оптимального маршруту
- підрахунок часу перебування на парковці
- моніторинг паркувальних місць
- статистика використання паркувальних місць

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Логічна структура програми складається з методів та функції, які виконують поставлені задачі. Мобільний додаток містить директорію компонентів, де знаходяться компоненти для користувацького інтерфейсу. Точкою входу для додатку є файл App з якого починається побудова дерева користувацького інтерфейсу. Серверний код організований за архітектурним паттерном MVC і складається з обробників де знаходиться уся бізнес логіка системи. Для роботи зі сховищем кожної сущності виділена директорія в якій знаходяться специфічні функції для роботи з- базою даних. Точкою входу для серверного додатку є файл index.ts в якому відбувається конфігурування системи та оголошення маршрутів. Також існують допоміжні модулі, які формують структуру програми — це конфіги, допоміжні функції тощо.

Усі модулі програми є незалежними та здатні до масштабування. Саме завдяки правильній побудові структури програми, вона є гнучкою, легкою в подальшій розробці та підтримці.

ТЕХНІЧНІ ЗАСОБИ ЩО ВИКОРИСТОВУЮТЬСЯ

Програмний код для мобільного додатку та для сервера розроблено у текстовому редакторі Visual Studio Code, для датчика програмний код розроблено в додатку Arduino IDE, на комп'ютері, що використовує операційну систему Fedora OS. Мовою програмування було обрано JavaScript та фреймворк React Native. Мобільний додаток написаний для операційної системи Android. Серверна частина додатку запускалася на хмарному провайдері під назвою Heroku.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Для повноцінного функціонування системи, паркувальне місце повинне бути обладнане датчиками, а також має бути доступний провідниковий інтернет. Для функціонування серверної програми на комп'ютері повинна бути встановлена NodeJS виже 12 версії. Для запуску додатку для телефона потрібно його встановити з файлу з розширенням apk. Телефон повинен мати доступ до інтернет і функцію отримання власної геопозиції.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними є інформація про парковки які вносяться обслуговуючим персоналом парковок.

Вихідними даними є список вільни, найближчих паркувальних місць до користувача, а також інформація про час перебування та час покидання паркувального місця.